MS&E 310 Project: Online Linear Programming

Michael Fairley

December 11, 2017

We consider the linear program:

$$
\begin{aligned}
&\max_x \sum_{j=1}^{n} \pi_j x_j && && (1)\\
&\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i && \forall i = 1, 2, \ldots, m && (2)\\
&\qquad 0 \leq x_j \leq 1 && \forall j = 1, 2, \ldots, n && (3)
\end{aligned}
$$

where $\pi_j \geq 0$ is the gain to allocate resources to bidder j, a_{ij} is the required quantity of resource i for bidder j and b_i is the total available quantity of resource i. We assume that $a_{ij} \in \{0, 1\}$.

In this project we consider the online version of this linear program, where $x_1, \ldots x_n$ are computed sequentially as $a_{i,1:m}$ is revealed. That is, bidders arrive sequentially and we must decide how much resource to allocate to the bidder before the next bidder arrives and we have no recourse on previous decisions.

The classical offline linear program provides an upper bound for the performance of the online linear program because the offline linear program has access to all of the information in the problem and can allocate resources to all bidders simultaneously. The offline linear program is feasible and bounded because $x = 0$ is always feasible and $\sum_{j=1}^{n} \pi_j$ is an upper bound for the objective function value. Therefore the offline linear program has an optimal solution.

1 Question 1

We consider the convex pari-mutuel call auction mechanism (CPCAM) model:

$$
\begin{aligned}
&\max_x \sum_{j=1}^{n} \pi_j x_j + u(s) && && (4)\\
&\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j + s_i = b_i && \forall i = 1, 2, \ldots, m && (5)\\
&\qquad 0 \leq x_j \leq 1 && \forall j = 1, 2, \ldots, n && (6)\\
&\qquad s_i \geq 0 && \forall i = 1, 2, \ldots, m && (7)
\end{aligned}
$$

The first-order KTT conditions for optimality are:
Stationarity:

$$\pi_j - \mu_{1j} + \mu_{2j} - \sum_{i=1}^{m} p_i a_{ij} = 0 \qquad \forall j = 1, 2, \ldots, n \tag{8}$$

$$\nabla_{s_i} u(s) + \mu_{3i} - p_i = 0 \qquad \forall i = 1, 2, \ldots, m \tag{9}$$

Complementary Slackness

$$\mu_{1j}(x_j - 1) = 0 \qquad \forall j = 1, 2, \ldots, n \tag{10}$$

$$\mu_{2j} x_j = 0 \qquad \forall j = 1, 2, \ldots, n \tag{11}$$

$$\mu_{3i} s_i = 0 \qquad \forall i = 1, 2, \ldots, m \tag{12}$$

Primal Feasibility

$$\sum_{j=1}^{n} a_{ij} x_j + s_i = b_i \qquad \forall i = 1, 2, \ldots, m \tag{13}$$

$$x_j - 1 \leq 0 \qquad \forall j = 1, 2, \ldots, n \tag{14}$$

$$-x_j \leq 0 \qquad \forall j = 1, 2, \ldots, n \tag{15}$$

$$-s_i \leq 0 \qquad \forall i = 1, 2, \ldots, m \tag{16}$$

Dual Feasibility

$$-\mu_{1j} \leq 0 \qquad \forall j = 1, 2, \ldots, n \tag{17}$$

$$-\mu_{2j} \leq 0 \qquad \forall j = 1, 2, \ldots, n \tag{18}$$

$$-\mu_{3i} \leq 0 \qquad \forall i = 1, 2, \ldots, m \tag{19}$$

where μ_{1j} is the dual variable for the constraint $x_j - 1 \leq 0$, μ_{2j} is the dual variable for the constraint $-x_j \leq 0$ and μ_{3i} is the dual variable for the constraint $-s_i \leq 0$.

The first-order KKT conditions are sufficient as the LP maximizes a concave function over a concave constraint set. The objective function is the sum of a linear function and strictly concave function and all constraints are linear.

We argue why the CPCAM model will have a unique solution for p. First, we note that since we are maximizing a strictly concave function with respect to s, there is a unique optimizer s^*.

We know that $\frac{\partial u(s)}{\partial s_i}_{s_i=0} = \infty$, so if $s_i = 0$, $p_i = \infty$ no matter the value of $\mu_{3i} \geq 0$. This provides a unique solution to p_i.

If $s_i > 0$ then $\mu_{3i} = 0$ and so $p_i = \nabla_{s_i} u(s^*)$. Therefore p_i is also unique.

From [2] and [1], s is the contingent amount of resource i that is kept by the market maker and $u(s)$ represents the “future value” of these contingent resources.

2 Question 2

We consider the following *online* optimization model:

$$\max_{x_k,s} \pi_k x_k + u(s) \tag{20}$$

$$\text{s.t.} \quad a_{ik}x_k + s_i = b_i - q_i^{k-1} \quad \forall i = 1,2,\ldots,m \tag{21}$$

$$0 \leq x_k \leq 1 \tag{22}$$

$$s_i \geq 0 \quad \forall i = 1,2,\ldots,m \tag{23}$$

where $q_i^{k-1} = \sum_{j=1}^{k-1} a_{ij}\bar{x_j}$ is the amount of resources i that have already been allocated to precedent bidders.

The KKT conditions are as follows:

Stationarity:

$$\pi_k - \mu_{1k} + \mu_{2k} - \sum_{i=1}^{m} p_i a_{ik} = 0 \tag{24}$$

$$\nabla_{s_i} u(s) + \mu_{3i} - p_i = 0 \quad \forall i = 1,2,\ldots,m \tag{25}$$

Complementary Slackness

$$\mu_{1k}(x_k - 1) = 0 \tag{26}$$

$$\mu_{2k}x_k = 0 \tag{27}$$

$$\mu_{3i}s_i = 0 \quad \forall i = 1,2,\ldots,m \tag{28}$$

Primal Feasibility

$$a_{ik}x_k + s_i = b_i - q_i^{k-1} \quad \forall i = 1,2,\ldots,m \tag{29}$$

$$x_k - 1 \leq 0 \tag{30}$$

$$-x_k \leq 0 \tag{31}$$

$$-s_i \leq 0 \quad \forall i = 1,2,\ldots,m \tag{32}$$

Dual Feasibility

$$-\mu_{1k} \leq 0 \tag{33}$$

$$-\mu_{2k} \leq 0 \tag{34}$$

$$-\mu_{3i} \leq 0 \quad \forall i = 1,2,\ldots,m \tag{35}$$

We now consider how this optimization problem can be solved efficiently. We assume that we have a solution to the bid $k-1$ with prices p^{k-1}. Then we have:

$$s^{k-1} = b - q^{k-2} - a_{k-1}x_{k-1} = b - q^{k-1} = a_k x_k + s \tag{36}$$

We now consider two high level cases. First, we consider if $\exists i : b_i - q_i^{k-1} = 0, a_{ik} = 1$. Then $x_k = 0$ is the only feasible solution and $s_i = 0$, $p_i = \infty$. Since $x_k = 0$, then for $\forall j\; s_j = s_j^{k-1}, p_j = \nabla_{s_i}(s^{k-1}) = p_j^{k-1}$.

Second, if $\forall i\, a_{ik} = 1 \Rightarrow b_i > 0$ then we consider three sub cases for values of x_k. If we can satisfy the KKT conditions under our assumptions for x_k, then we have found an optimal solution.

2.1 $x_k = 0$

We know that $x_k = 0$ is always feasible, so all that remains is to check the pricing conditions. We know that the prices will remain the same because $s = s^{k-1}$, so if $\pi_k \leq \sum_i^m p_i^{k-1} a_{ik}$ then $x_k = 0$ is optimal.

2.2 $x_k = 1$

We must check if there are enough resources remaining and then check the pricing conditions. So if $s = b - q^{k-1} - a_k \geq 0$ and $\pi_k \geq \sum_i^m p_i a_{ik}$ where $p_i = \nabla_{s_i} u(b - q^{k-1} - a_k)$ then $x_k = 1$ is optimal.

2.3 $0 < x_k < 1$

We have that $\pi_k = \sum_i^m p_i a_{ik}$, so we can find the root of the following function, which is a function of only x_k:

$$f(x) = \pi_k - \sum_i^m \nabla_{s_i} u(b - q_i^{k-1} - a_{ik} x_k) \tag{37}$$

We can use Newton's method to find the root.

3 Question 3

We ran an experiment to test the convergence of the online CPCAM model under two different utility functions. We found that the prices of all goods remained near zero until the resources ran out. This makes sense as the prices are the shadow prices for the resources, indicating how much it is worth to the market maker to have more resources. When there is a surplus of resources at the beginning, there is no value in having additional resources so the prices are near zero. We found that the prices do not converge (Figure 1) to the grand truth under any of the utility functions, but that using u_2 with $w = 1$ causes the prices to become closer to the grand truth, while all other utility functions cause the prices to diverge from the grand truth. Despite this, the prices do stabilize because when the resources are very low, the bid amount required to complete an order is very high and so most bids are rejected and the prices remain the same as the previous time step.

We show an example of the price change for good 1 for each of the utility functions in Figure 2. In Appendix A we provide further plots of the prices of all goods. We note that the prices are non-decreasing.

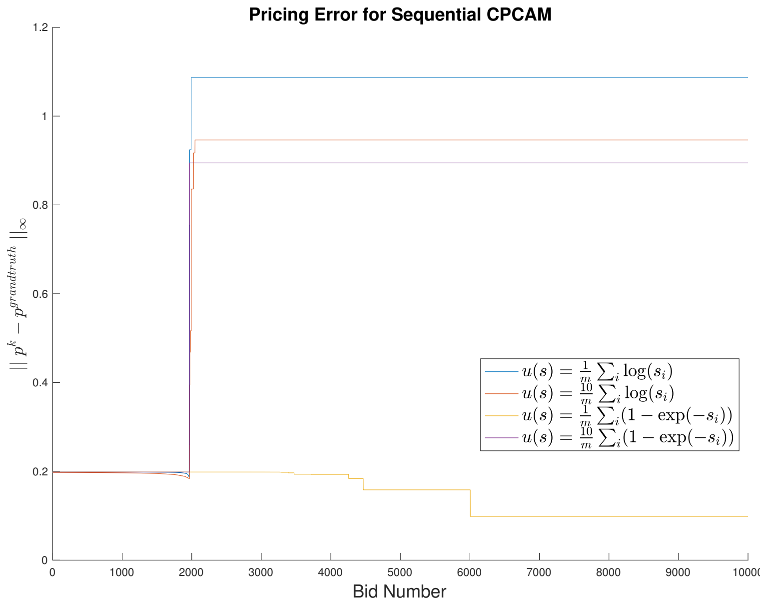

Figure 1: Pricing Error for Online CPCAM

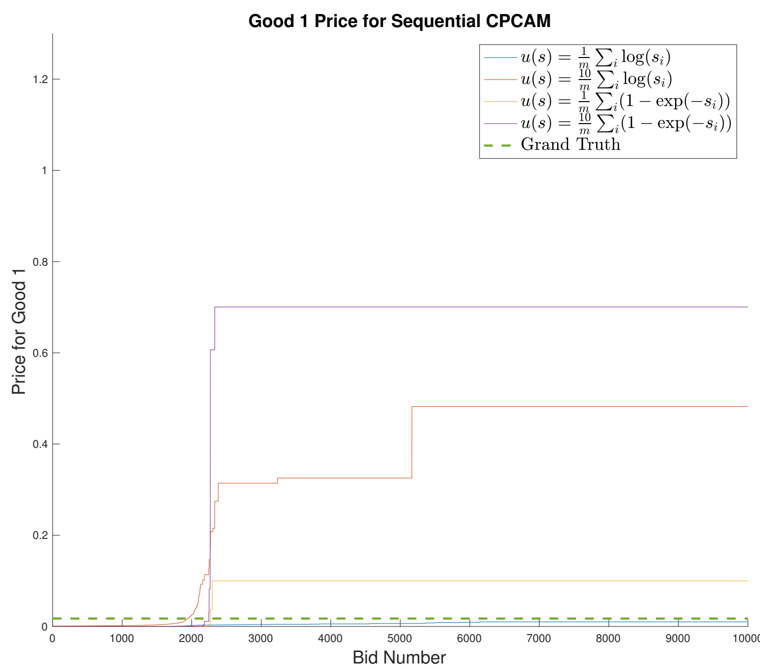

Figure 2: Price of Good 1 for Online CPCAM

4 Question 4 and 5

We ran an experiment to measure the performance of the online SLPM optimization model. We found that the higher the value of k, the closer the performance of the online algorithm to the offline solution. Dynamic updating of the prices at time points determined by a geometric series performed even closer to the optimal solution. An example run of the models is shown in Figure 3. We note that the optimal solution uses the resources at a constant rate over the time horizon, whereas the online models tend to use up the resources before the end of the time horizon. In Table 1, we give confidence intervals for 100 runs of the bidding process.

We also consider the price stability for the geometric series and we notice that the price is not non-decreasing (Figure 4), in contrast to the online CPCAM but the prices do converge closer to the grand truth than online CPCAM.

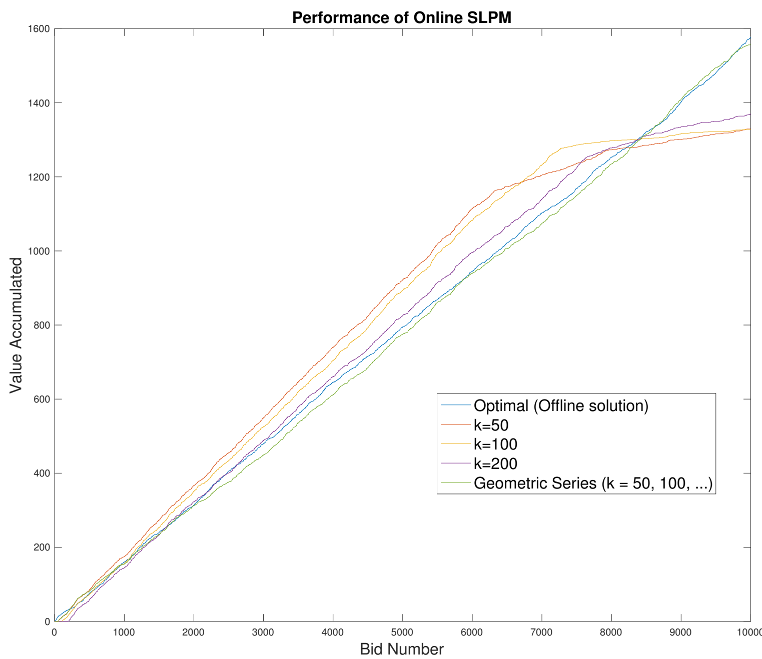

Figure 3: Performance of SLPM models under different k values

k	95% CI for Simulated Competitive Ratio
50	0.7659 [0.7569, 0.7749]
100	0.8100 [0.8019, 0.8181]
200	0.8577 [0.8507, 0.8647]
Geometric (50, 100, ...)	0.9684 [0.9665, 0.9703]

Table 1: Competitive ratio different values of k

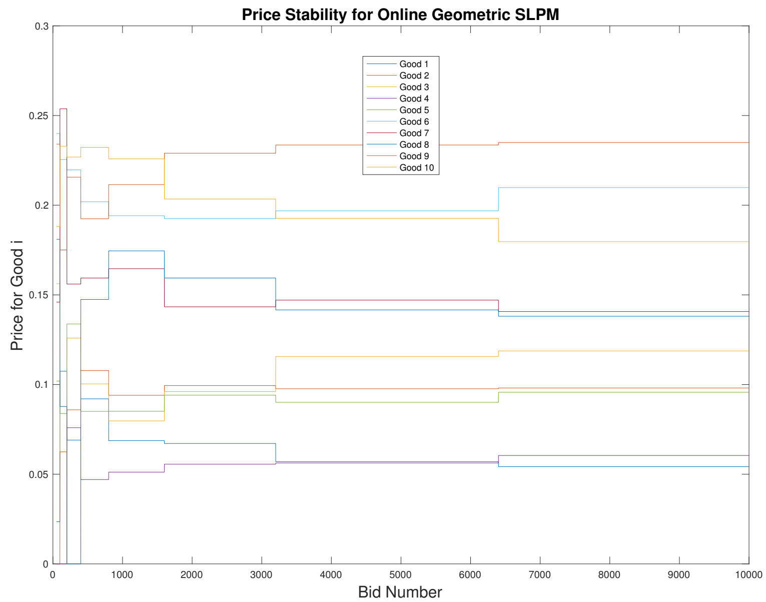

Figure 4: Prices of goods for geometric SLPM

5 Question 6

We now consider an extension to the resource allocation problem where there are production costs.

$$\max_x \sum_{j=1}^{n} (\pi_j x_j - \sum_{i=1}^{m} \sum_{k=1}^{K} c_{ijk} y_{ijk}) \tag{38}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} y_{ijk} = a_{ij} x_j \qquad \forall i, j \tag{39}$$

$$\sum_{i,j} y_{ijk} \leq b_k \qquad \forall k = 1, 2, \ldots, K \tag{40}$$

$$0 \leq x_j \leq 1 \qquad \forall j = 1, 2, \ldots, n \tag{41}$$

$$y_{ijk} \geq 0 \qquad \forall i, j, k \tag{42}$$

where c_{ijk} is the cost to allocate good i, which is produced by producer $k = 1, \ldots, K$ to bidder j.

The dual of this linear program can be written as:

$$
\begin{aligned}
&\min_{\lambda,p,\mu} \sum_{k=1}^{K} b_k p_k + \sum_{j=1}^{n} \mu_j && (43)\\
&\text{s.t.} \quad \mu_j + \sum_{i=1}^{m} a_{ij}\lambda_{ij} \geq \pi_j \quad \forall j = 1, \dots n && (44)\\
&\qquad \lambda_{ij} - p_k \leq c_{ijk} \qquad \forall i, j, k && (45)\\
&\qquad p_k \geq 0 \qquad \forall k = 1, 2, \dots, K && (46)\\
&\qquad \mu_j \geq 0 \qquad \forall j = 1, 2, \dots, n && (47)\\
&\qquad \lambda_{ij} \quad \text{free} \qquad \forall i, j, k && (48)
\end{aligned}
$$

and by complementary slackness we have

$$
\begin{aligned}
p_k(\sum_{i,j} y_{ijk} - b_k) &= 0 && \forall k = 1, \dots, K && (49)\\
\mu_j(x_j - 1) &= 0 && \forall j = 1, \dots, n && (50)\\
x_j(\mu_j + \sum_{i=1}^{m} a_{ij}\lambda_{ij} - \pi_j) &= 0 && \forall j = 1, \dots, n && (51)\\
y_{ijk}(p_k + c_{ijk} - \lambda_{ij}) &= 0 && \forall i, j, k && (52)
\end{aligned}
$$

We can see from these conditions that the value of x_j implies similar pricing conditions to the classical problem. A key difference is that the pricing of the good is dependent on the bid number, j, because the cost of producing that good also depends on j.

$$x_j = 0 \Rightarrow \pi_j \leq \sum_{i}^{m} \lambda_{ij} a_{ij} \tag{53}$$

$$x_j = 1 \Rightarrow \pi_j \geq \sum_{i}^{m} \lambda_{ij} a_{ij} \tag{54}$$

$$0 < x_j < 1 \Rightarrow \pi_j = \sum_{i}^{m} \lambda_{ij} a_{ij} \tag{55}$$

We also see that $y_{ijk} > 0 \Rightarrow p_k + c_{ijk} - \lambda_{ij} = 0$ Then if $\exists k_1, k_2 : y_{ijk_1} > 0, y_{ijk_2} > 0 \Rightarrow p_{k_1} + c_{ijk_1} = p_{k_2} + c_{ijk_2}$.

We can use these conditions to develop an online algorithm. We take a similar approach to SLPM where we do not accept any bids for the first h iterations and we use a one-shot learning linear program with the resources set to $\frac{h}{n} b_k$. This gives us a sampled value for p_k.

For subsequent bids, we use p_k to solve a sub-optimization problem for the least cost production plan for a bid and then we check if the bid is competitive enough given the cost of the production plan. This amounts to estimating λ_{ij} via an optimization to check if Equation (54) holds. If (54) does not hold, then we do not accept the bid. If there is no feasible production plan for the bid due to resource constraints, then we also do not accept the bid.

The sub-optimization problem at iteration l to generate a production plan is:

$$\begin{aligned}
&\min_{\lambda_{il}, y_{ilk}} \sum_{i}^{m} \lambda_{il} a_{il} \\
&\text{s.t.} \qquad \lambda_{il} = \sum_{k}^{K} y_{ilk}(p_k + c_{ilk}) && \forall i = 1, 2, \ldots, m \\
&\qquad\qquad \sum_{k=1}^{K} y_{ilk} = a_{il} && \forall i = 1, 2, \ldots, m \\
&\qquad\qquad \sum_{i} y_{ilk} \leq b_k - q^{l-1} && \forall k = 1, 2, \ldots, K \\
&\qquad\qquad y_{ilk} \in \{0, 1\} && \forall i, k
\end{aligned}$$

Since y_{ikj} is integer, exactly one supplier, k^* will generate the value of λ_{il} under the first constraint and so $\lambda_{il} = p_{k^*} + c_{ilk^*}$, which is consistent with form of λ_{ij} for $j = l$ in the full optimization problem. Future work will involve checking if the relaxation of this integer program is naturally integer.

After solving this sub-optimization problem, we let $x_l = 1$ if

$$\pi_l > \sum_{i}^{m} \lambda_{il} a_{ij} \tag{56}$$

If the sub-optimization problem is infeasible or (56) does not hold then we set $x_l = 0$

The KKT conditions (excluding primal feasibility, which is given above) for the relaxation of the sub-optimization problem are:

Stationarity:

$$-a_{il} - t_i = 0 \qquad \forall i = 1, \ldots, m \tag{57}$$

$$u_i(p_k + c_{ilk}) + v_i + s_k - \mu_{1ik} + \mu_{2ik} = 0 \qquad \forall i, k \tag{58}$$

Complementary Slackness:

$$s_k(\sum_i y_{ilk} - b_k - q^{l1}) = 0 \qquad \forall k = 1, \ldots, K \tag{59}$$

$$\mu_{1ik} y_{ilk} = 0 \qquad \forall i, k \tag{60}$$

$$\mu_{2ik}(y_{ilk} - 1) = 0 \qquad \forall i, k \tag{61}$$

Dual Feasibility:

$$s_k \geq 0 \qquad \forall k = 1, \ldots, K \tag{62}$$

$$\mu_{1ik} \geq 0 \qquad \forall i, k \tag{63}$$

$$\mu_{2ik} \geq 0 \qquad \forall i, k \tag{64}$$

I ran this algorithm on simulated bidding data with different values of h using the linear programming relaxation of the sub-optimization integer program. I also ran a "greedy" algorithm with $h = 50$ that solves the sub-optimization problem at each step but does not check for price competitiveness, so it accepts every bid that is feasible to accept.

The data generating process is defined in the Julia code in Appendix C lines 6 through 39. The specific instance of values for the plots below was:
b = [2500.0, 2500.0]
Mean Cost Per Item, Supplier
Supplier 1, mean item cost = [1.94286, 2.43703, 5.45992, 9.22667, 3.27307, 2.06395, 9.11909, 4.26143, 8.43837, 9.65688]
Supplier 2, mean item cost = [4.83836, 7.07352, 8.31443, 6.89239, 7.05998, 5.27056, 3.07869, 6.42893, 4.44837, 8.03415]
Mean Bid Pricing = [3.13844, 4.5031, 6.63501, 7.80736, 4.91436, 3.41509, 5.84672, 5.09301, 6.1912, 8.59334]

We note that $h = 100$ performs worse than $h = 50$, which differs from the pattern in SLPM where the higher h (denoted k in SLPM), the better the performance. Future work would involve determining how to generate random bidding data in a way that we can better reason about the optimal performance (in a way analogous to using a grand truth price vector), determining the conditions under which this algorithm has a specific competitive ratio and how h impacts performance under these conditions.

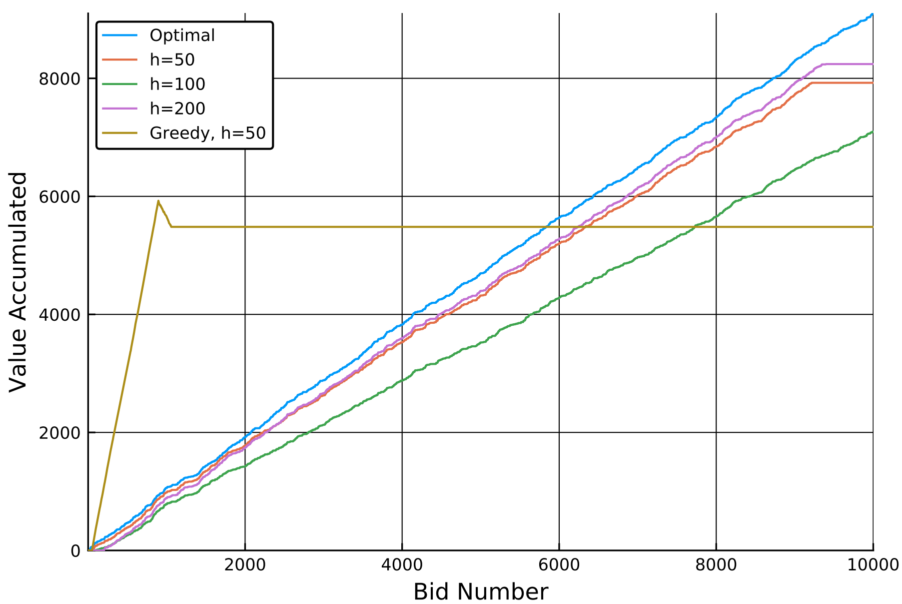

Figure 5: Performance of online resource allocation with production costs algorithm

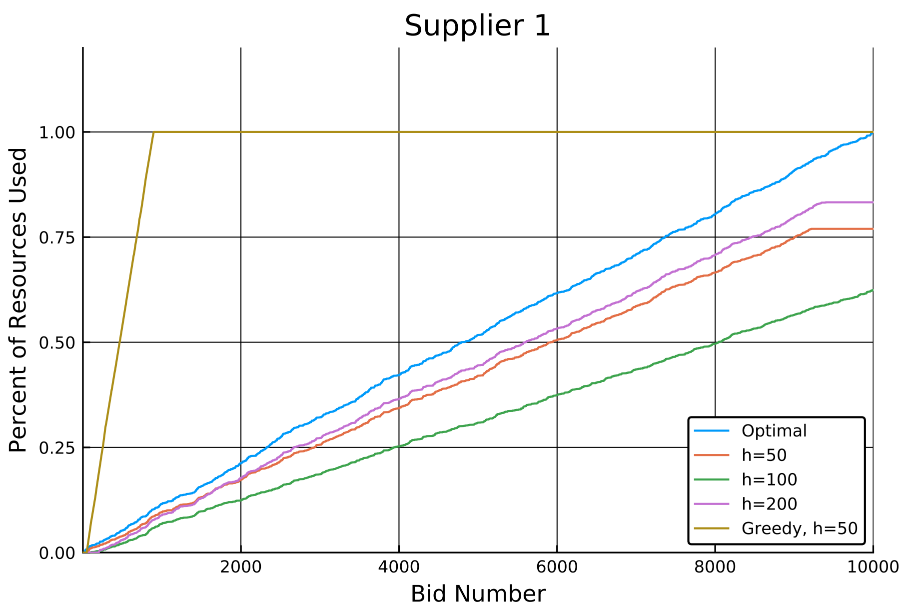

Figure 6: Percentage of supplier 1's resources used over time

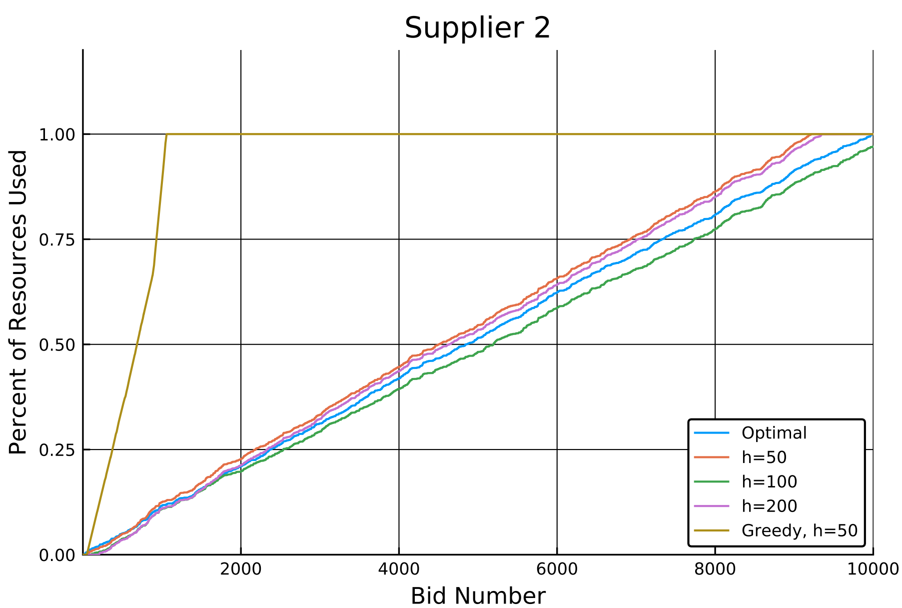

Figure 7: Percentage of supplier 2's resources used over time

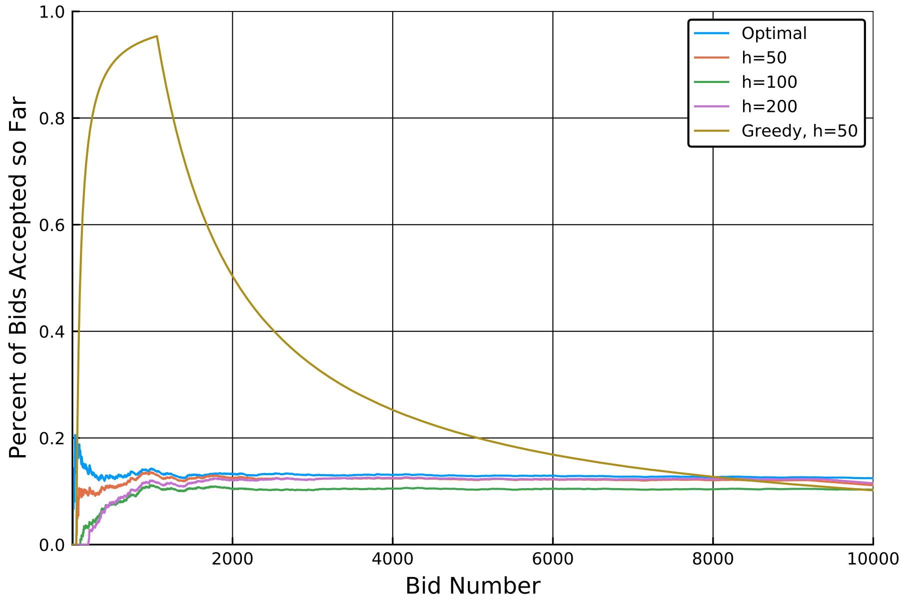

Figure 8: Percentage of bids accepted so far

References

[1] Shipra Agrawal et al. "A Unified Framework for Dynamic Prediction Market Design". In: *Operations Research* 59.3 (2011), pp. 550–568.

[2] Mark Peters, Anthony Man-Cho So, and Yinyu Ye. "Pari-mutuel Markets: Mechanisms and Performance". In: *Proceedings of the 3rd International Conference on Internet and Network Economics*. 2007.

A Price of Goods in Online CPCAM

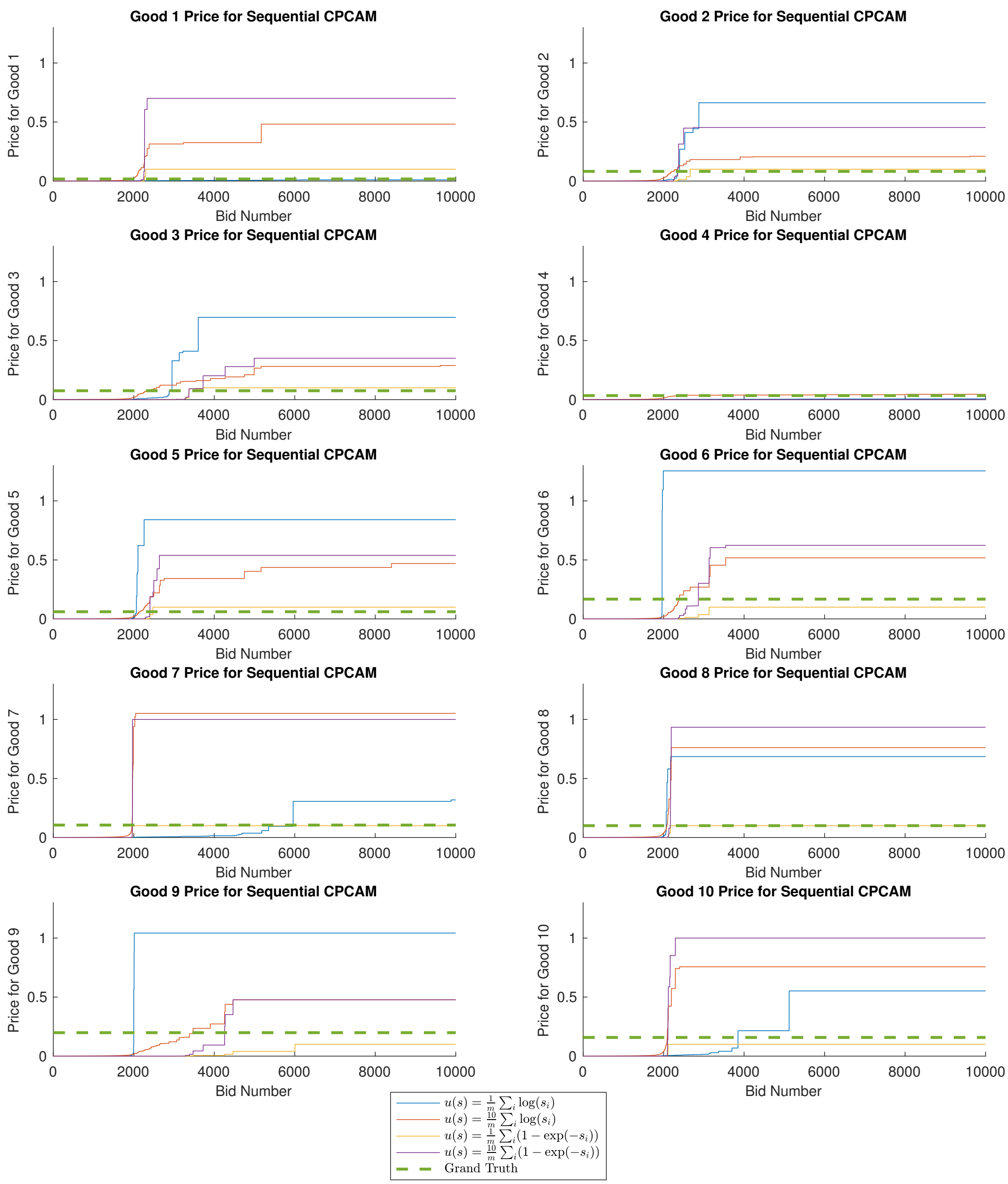

Figure 9: Pricing for Online CPCAM

B MATLAB Code for Online CPCAM and SLPM

```matlab
function [X, prices] = solve_scpm(n, m, b, bid_generator, obj_fun, x0)
options = optimoptions('fmincon', 'SpecifyObjectiveGradient', true,...
    'display', 'off');
q = zeros(m, 1);
prices = zeros(m, n);
X = zeros(n, 1);
for j = 1:n
    [a_k, pi_k] = bid_generator();

    if j > 1 && (pi_k <= prices(:, j-1)'*a_k || ~all(b(a_k==1) - q(a_k==1) > 0))
        X(j) = 0.0;
        prices(:, j) = prices(:, j-1);
    else
        s1 = b - q - a_k;
        [~, g] = obj_fun([1; s1], pi_k);
        p1 = -g(2:end);
        if all(s1 >= 0) && (pi_k >= p1'*a_k)
            X(j) = 1.0;
            prices(:, j) = p1;
        else
            fun = @(xs) obj_fun(xs, pi_k);
            [xs, ~, ~, ~, ~, ~, ~] = fmincon(fun, x0, [], [],...
                [a_k, eye(m)], max(b-q, 0), zeros(m+1, 1),...
                [1; Inf(m, 1)], [], options);
            [~, g] = obj_fun(xs, pi_k);
            prices(:, j) = -g(2:end);
            X(j) = xs(1);
        end
    end
        q = q + a_k * X(j);

    if mod(j, 100) == 0
        fprintf('Solved iteration %i\n', j)
    end

end
return
```

```matlab
function [X, prices, value] = online_slpm(n, k_vector, m, b, bid_generator)
options = optimoptions('linprog', 'display', 'off');
q = zeros(m, 1);
A = zeros(m, n);
PI = zeros(n, 1);
prices = zeros(m, n);
X = zeros(n, 1);
value = zeros(n + 1, 1);
```

```

prices(:, 1) = Inf;
ki = 1;
k = k_vector(ki);

for j = 1:n
    [a_k, pi_k] = bid_generator();
    A(:, j) = a_k;
    PI(j) = pi_k;

    if pi_k > prices(:, j)' * a_k && all(a_k <= b - q)
        X(j) = 1.0;
    else
        X(j) = 0.0;
    end
    value(j+1) = value(j) + X(j) * pi_k;
    q = q + X(j) * a_k;

    if j == k
        if ki < length(k_vector)
            ki = ki + 1;
            k = k_vector(ki);
        end
        [~, ~, ~, ~, lambda] = linprog(-PI(1:j), A(:, 1:j), (j / n) * b,...
            [], [], zeros(j, 1), ones(j, 1), options);
        prices(:, j+1) = lambda.ineqlin;
    elseif j < n
        prices(:, j+1) = prices(:, j);
    end
end
end
```

C Julia Code for Online Production Problem

```
using JuMP
using Gurobi

const output_flag = 0

function problem_setup(m, K, seed; total_resource=500.0, u = 1, v = 10)
    rng_gt = MersenneTwister(seed + 200)

    b = ones(K) * m * total_resource / K
    # Assume cost is normally distributed around a mean_ik for each good
    ↪  and supplier
    # pre compute costs
    C_mean = u + (v - u) * rand(rng_gt, m, K)
```

```julia
    C = randn(rng_gt, m, n, K)
    for j = 1:n
        C[:, j, :] = C_mean + C[:, j, :] * 0.2
    end
    # r_c_mean = u + (v - u) * rand(rng_gt, m)
    r_c_mean = mean(C_mean, 2) + randn(rng_gt) * 0.2
    temp = r_c_mean
    r_c_mean = zeros(m)
    for i = 1:m
        r_c_mean[i] = temp[i]
    end

    println("Problem:")
    println("b = $b")
    println("Mean Cost Per Item, Supplier")
    for k = 1:K
        println("Supplier $k, mean item cost = $(C_mean[:, k])")
    end
    println("Mean Bid Pricing = $r_c_mean")

    rng = MersenneTwister(seed)
    bid_generator = function () return rand(rng, 0:1, m) end
    c_generator = function (j) return C[:, j, :] end
    pi_generator = function (a_k) return transpose(r_c_mean) * a_k + randn(rng) * 0.2 end

    return b, bid_generator, c_generator, pi_generator, rng
end

function offline_lp(env, n, m, K, b, bid_generator::Function, c_generator::Function, pi_generator::Function)
    # collect all data first
    q = zeros(n + 1, K) # resources used
    A = zeros(m, n)
    C = zeros(m, n, K)
    π = zeros(n)
    value = zeros(n + 1)

    for j = 1:n
        A[:, j] = bid_generator()
        C[:, j, :] = c_generator(j)
        π[j] = pi_generator(A[:, j])
    end

    _, prices, X, Y, _ = solve_primal(env, n, m, K, b, A, C, π)

    for j = 1:n
```

```
        value[j + 1] = value[j] + X[j] * π[j] - sum(Y[:, j, :] .* C[:, j, :])
        for k = 1:K
            q[j + 1, k] = q[j, k] + sum(Y[:, j, k])
        end
    end

    return prices, X, value, q

end

function online_lp(env, n, h_vector, m, K, b, bid_generator::Function, c_generator::Function, pi_generator::Function;
    greedy=false, print_every=100)
    q = zeros(n + 1, K) # resources used
    A = zeros(m, n)
    C = zeros(m, n, K)
    π = zeros(n)
    prices = zeros(K, n)
    X = zeros(n)
    Y = zeros(m, n, K)
    value = zeros(n + 1)

    # initialize price so that first k bids are not fulfilled
    h = shift!(h_vector)
    h1 = h
    for j = 1:n
        # draw bid
        A[:, j] = bid_generator()
        C[:, j, :] = c_generator(j)
        π[j] = pi_generator(A[:, j])

        # solve ip to get y
        if j >= h1
            feasible, y, obj = ip_sub(env, m, K, b - q[j, :], A[:, j], C[:, j, :], prices[:, j])
            if feasible == true && (π[j] > obj || greedy == true)
                X[j] = 1.0
                Y[:, j, :] = y
            else
                X[j] = 0.0
                Y[:, j, :] = 0
            end
        else
            X[j] = 0.0
            Y[:, j, :] = 0
        end

```

```julia
        for k = 1:K
            q[j + 1, k] = q[j, k] + sum(Y[:, j, k])
        end
        value[j + 1] = value[j] + X[j] * π[j] - sum(Y[:, j, :] .* C[:, j, :])

        # learning step
        if j == h && j < n
            if !isempty(h_vector) h = shift!(h_vector) end
            _, prices[:, j + 1], _, _, _ = solve_primal(env, j, m, K, (b - q[j, :]) * (j / n), A[:, 1:j], C[:, 1:j, :], π[1:j])
        elseif j < n
            prices[:, j + 1] = prices[:, j]
        end
    end
    return prices, X, value, q

end

function solve_primal(env, n, m, K, b, A, C, π)
    model = Model(solver=GurobiSolver(env, OutputFlag = 0))
    @variable(model, 0 <= x[1:n] <= 1)
    @variable(model, y[1:m, 1:n, 1:K] >= 0)

    resource_constraint = []
    for i = 1:m
        for j = 1:n
            push!(resource_constraint, @constraint(model, sum(y[i, j, :]) == A[i, j] * x[j]))
        end
    end

    production_constraint = []
    for k = 1:K
        push!(production_constraint, @constraint(model, sum(y[:, :, k]) <= b[k]))
    end

    @objective(model, Max, sum(x .* π) - sum(C .* y))

    status = solve(model)

    if status == :Optimal
        return getdual(resource_constraint), getdual(production_constraint), getvalue(x), getvalue(y), getobjectivevalue(model)
    else
        print(model)
```

```
        error("No production prices found. Status = $status")
    end
end

function solve_dual(env, n, m, K, b, A, C, π)
    model = Model(solver = GurobiSolver(env, OutputFlag=0))
    @variable(model, λ[1:m, 1:n])
    @variable(model, p[1:K] >= 0)
    @variable(model, μ[1:n] >= 0)

    c1 = []
    for j = 1:n
        push!(c1, @constraint(model, μ[j] + sum(A[:, j] .* λ[:, j]) >= π[j]))
    end

    c2 = []
    for i = 1:m
        for j = 1:n
            for k = 1:K
                push!(c2, @constraint(model, λ[i, j] - p[k] <= C[i, j, k]))
            end
        end
    end

    @objective(model, Min, sum(b .* p) + sum(μ))

    status = solve(model)

    if status == :Optimal
        return getdual(c1), getdual(c2), getvalue(λ), getvalue(p), getvalue(μ), getobjectivevalue(model)
    else
        print(model)
        error("No optimal solution to dual found. Status = $status")
    end
end

function ip_sub(env, m, K, b, A, C, p)
    model = Model(solver=GurobiSolver(env, OutputFlag=output_flag))
    @variable(model, λ[1:m])
    @variable(model, 0 <= y[1:m, 1:K] <= 1)

    for i = 1:m
        @constraint(model, λ[i] == sum(y[i, :] .* (p + C[i, :])))
    end

    for i = 1:m
```

```
        @constraint(model, sum(y[i, :]) == A[i])
    end

    for k = 1:K
        @constraint(model, sum(y[:, k]) <= b[k])
    end

    @objective(model, Min, sum(λ .* A))

    status = solve(model, suppress_warnings=true)

    if status == :Optimal
        return true, getvalue(y), getobjectivevalue(model)
    elseif status == :Infeasible
        return false, zeros(m, K), Inf
    else
        print(model)
        error("Subproblem not optimal or infeasible. Status = $status")
    end
end

function simulation_q6(n, m, K; seed=1234)
    b, bg, cg, pig, rng = problem_setup(m, K, seed)

    env = Gurobi.Env()

    prices = zeros(K, n, 5)
    X = zeros(n, 5)
    value = zeros(n + 1, 5)
    q = zeros(n + 1, K, 5)

    # offline lp
    srand(rng, seed)
    println("Computing offline LP")
    prices_opt, X[:, 1], value[:, 1], q[:, :, 1] = offline_lp(env, n, m, K, b, bg, cg, pig)
    for k = 1:K
        prices[k, :, 1] = prices_opt[k]
    end

    # online lp with k_vector
    h_vector = [50]
    srand(rng, seed)
    println("Computing online LP, h = 50")
    prices[:, :, 2], X[:, 2], value[:, 2], q[:, :, 2]  = online_lp(env, n, h_vector, m, K, b, bg, cg, pig)

    h_vector = [100]
```

```
    srand(rng, seed)
    println("Computing online LP, h = 100")
    prices[:, :, 3], X[:, 3], value[:, 3], q[:, :, 3]  = online_lp(env, n,
    ↪  h_vector, m, K, b, bg, cg, pig)

    h_vector = [200]
    srand(rng, seed)
    println("Computing online LP, h = 200")
    prices[:, :, 4], X[:, 4], value[:, 4], q[:, :, 4]  = online_lp(env, n,
    ↪  h_vector, m, K, b, bg, cg, pig)

    # greedy
    h_vector = [50]
    srand(rng, seed)
    println("Computing online greedy LP, h = 50")
    prices[:, :, 5], X[:, 5], value[:, 5], q[:, :, 5]  = online_lp(env, n,
    ↪  h_vector, m, K, b, bg, cg, pig, greedy=true)

    return b, prices, X, value, q
end
```